FIG. 1A

```
<book>
   <title> XML </title>
   <allauthors>
      <author> jane </author>
      <author> john </author>
   </allauthors>
   <year> 2000 </year>
   <chapter>
      <head> Origins </head>
      <section>
         <head> ...</head>
         <section> ...</section>
      </section>
      <section> ...</section>
   </chapter>
   <chapter> ...</chapter>
</book>
```

FIG. 1B

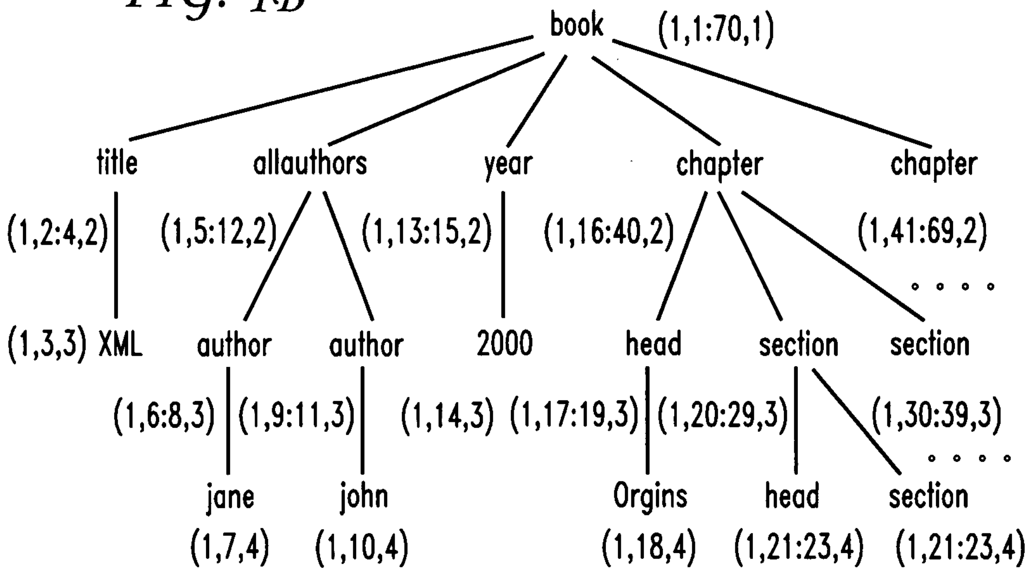


FIG. 2A

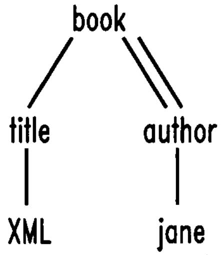


FIG. 2B

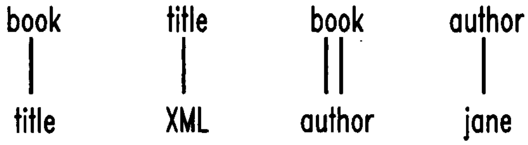


FIG. 3

```
Algorithm Tree-Merge-Anc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* Alist is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

begin-desc = DList->firstNode; OutputList = NULL;
for (a = AList->firstNode; a ! = NULL; a = a->nextNode) {
   for (d = begin-desc; (d ! = NULL && d.StartPos < a.StartPos) ; d = d->nextNode) {
      /* skipping over unmatchable d's */ }
   begin-desc = d:
   for (d = begin-desc; (d ! = NULL && d.EndPos < a.EndPos); d = d->nextNode) {
      if ( (a.StartPos < d.StartPos) && (d.EndPos < a.EndPos)
             [&& (d.LevelNum = a.LevelNum + 1)]) {
         /* the optional condition is for parent-child relationship */
         append (a,d) to OutputList; }
   }
}
```

FIG. 4

```
Algorithm tree-merge-Desc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* Alist is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

begin-anc = AList->firstNode; OutputList = NULL;
for (d = DList->firstNode; d != NULL; d = d->nextNode) {
   for (a = begin-anc; (a != NULL && a.EndPos < d.StartPos); a = a->nextNode) {
      /* skipping over unmatchable a's */ }
   begin-anc = a;
   for (a = begin-anc; (a != NULL && a.StartPos); a = a->nextNode) {
      if ( (a.StartPos < d.StartPos) && (d.EndPos < a.EndPos)
             [&& (d.LevelNum = a.LevelNum + 1) ] ) {
         /* the optional condition is for parent-child relationships */
         append (a,d) to OutputList; }
   }
}
```

FIG. 5A

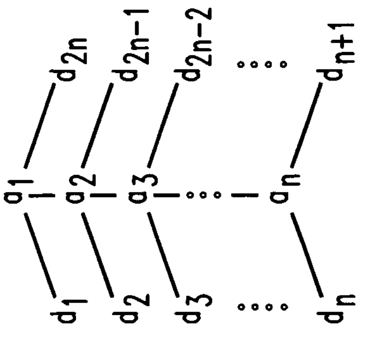


FIG. 5B

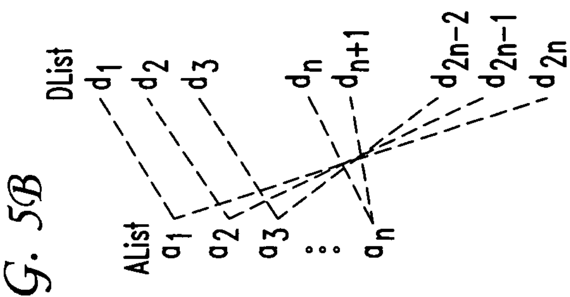


FIG. 5C

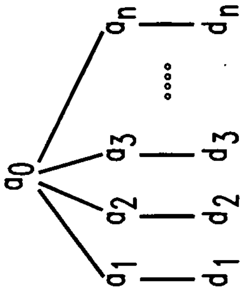


FIG. 5D

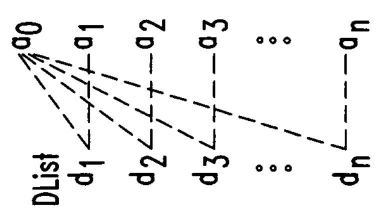


FIG. 6

```
Algorithm Stack-Tree-Desc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descentants in sorted order of StartPos */

a = AList->firstNode; d = Dlist->firstNode; OutputList = NULL;
while (the input list are not empty or the stack is not empty) {
   if ( (a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos) ) {
      /* time to pop the top element in the stack */
      tuple = stack->pop(); }
   else if (a.StartPos < d.StartPos) {
      stack->push (a)
      a = a->nextNode }
   else {
      for (al = stack->bottom; al != NULL; al = al->up) {
         append (al,d) to OutputList
      }
      d = d->nextNode
   }
}
```

FIG. 7A

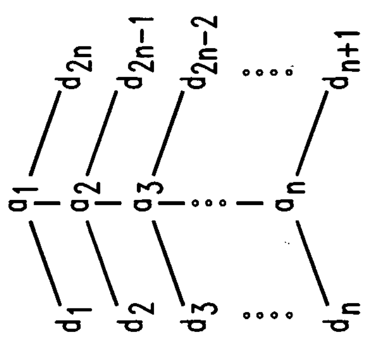


FIG. 7B

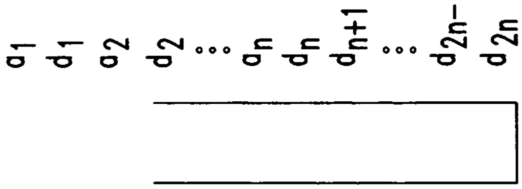


FIG. 7C

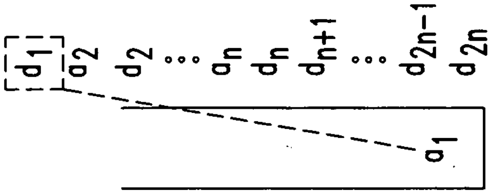


FIG. 7D

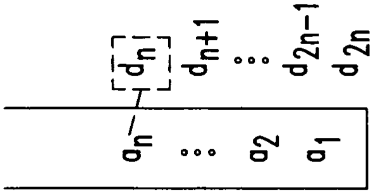


FIG. 7E

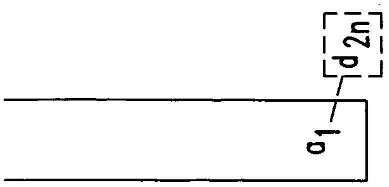


FIG. 8

```
Algorithm Stack-Tree-Anc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descentants in sorted order of StartPos */

a = AList->firstNode; d = Dlist->firstNode; OutputList = NULL;
while (the input list are not empty or the stack is not empty) {
   if ( (a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos) ) {
      /* time to pop the top element in the stack */
      tuple = stack->pop(); }
      if (stack->size == 0) { /* we just popped the bottom element */
         append tuple.inherit-list to OutputList }
      else  {
         append tuple.inherit-list to tuple.self-list
         append the resulting tuple.self-list to stack->top.inherit-list
      }
   }
   else if (a.StartPos < d.StartPos) {
      stack->push (a)
      a = a->nextNode }
   else {
      for (al = stack->bottom; al != NULL; al = al->up) {
         if (al == stack->bottom) append (al,d) to OutputList
         else append (al,d) to the self-list of al
      }
      d = d->nextNode
   }
}
```

FIG. 9

```
<!ELEMENT manager (name, (manger | department | employee)+)>
<!ATTLIST manager id CDATA #FIXED "1">
<!ELEMENT department (name, email?, employee+, department*)>
<!ATTLIST department id CDATA #FIXED "2">
<!ELEMENT employee (name+, email?)>
<!ATTLIST employee id CDATA #FIXED "3">
<!ELEMENT name (#PCDATA)>
<!ATTLIST name id CDATA #FIXED "4">
<!ELEMENT email (#PCDATA)>
<!ATTLIST email id CDATA #FIXED "5">
```

FIG. 9A

Node	Count
manager	25,880
departmaent	342,450
employee	574,530
email	250,530

FIG. 9B

Query	XQuery Path Expression	Result Cardinality
QS1	employee/email	140,700
QS2	employee//email	142,958
QS3	manger/department	16,855
QS4	manager//department	587,137
QS5	manager/employee	17,259
QS6	manager//employee	990,774
QC1	manager/employee/email	7,990
QC2	manager//employee/email	232,406

FIG. 10

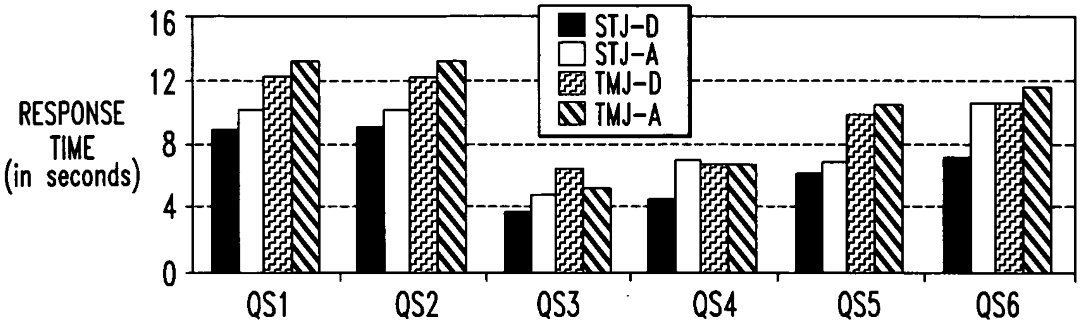
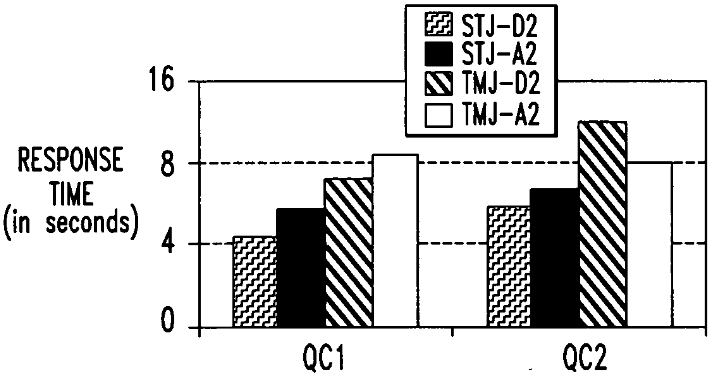

16
12
8
4
0
RESPONSE TIME (in seconds)
■ STJ-D
□ STJ-A
▨ TMJ-D
▧ TMJ-A
QS1
QS2
QS3
QS4
QS5
QS6

FIG. 11


▨ STJ-D2
■ STJ-A2
▧ TMJ-D2
□ TMJ-A2
16
8
4
0
RESPONSE TIME (in seconds)
QC1
QC2